

# JEGYZŐKÖNYV

Mesterséges intelligencia és neurális hálózatok  
Féléves beadandó – Genetikus algoritmus

Készítette: **Orosz Kristóf**  
Neptunkód: **EYZWG9**  
Dátum: 2025. november 27.

**Sárospatak, 2025**

## Tartalomjegyzék

Bevezetés .....	3
A projekt genetikus algoritmusá .....	3

## Bevezetés

A féléves beadandómban egy genetikus algoritmust valósítottam meg, amellyel több e-ollereket tartalmazó depók között keresem a lehető legrövidebb útvonalat. Első lépésként létrehoztam egy függvényt, amellyel kiszámítom két raktár közötti euklideszi távolságot a koordinátáik alapján. Végigmegy az algoritmus a raktárak sorrendjén, megkeresi a hozzájuk tartozó pontokat, majd összeadja az egymást követő helyek közötti távolságokat.

A genetikus algoritmusom először véletlenszerű útvonalakból álló kezdő populációt hoz létre. Ehhez a Fisher–Yates keverést használom, hogy minden lehetséges sorrend egyenlő eséllyel fordulhasson elő. Miután létrejött a populáció, rendezi az egyedeket a teljes útvonalhosszuk alapján, így az élre kerülnek azok a megoldások, amelyek a legrövidebb utat adják. Ezeket tekinti a következő generáció lehetséges szülőinek.

A genetikus rész két fontos része a keresztezés és a mutáció. A keresztezés során két útvonalból próbál összeállítani egy új, jobb megoldást: az egyik szülőből átmásol egy részzszakaszt, a maradék helyeket pedig a másik szülő sorrendje szerint tölti ki. A genetikus ciklust több generáción át ismétli az algoritmusom, minden körben a legjobb megoldásokból létrehozva az új populációt.

A folyamat végén kiválasztja az aktuális generáció legjobb útvonalát, majd kiszámítja a hozzá tartozó teljes távolságot.

## A projekt genetikus algoritmus

```
// Euklideszi távolság kiszámítása két pont között
function tavolsag(a, b) {
  return Math.sqrt((a.x - b.x) ** 2 + (a.y - b.y) ** 2);
}

// Egy útvonal teljes hossza
function utHossz(ut, raktarak) {
  let hossz = 0;
  for (let i = 0; i < ut.length - 1; i++) {
    const a = raktarak.find(d => d.nev === ut[i]);
    const b = raktarak.find(d => d.nev === ut[i + 1]);
    hossz += tavolsag(a, b);
  }
  // Visszatérés az induló raktárba
  const elso = raktarak.find(d => d.nev === ut[0]);
  const utolso = raktarak.find(d => d.nev === ut[ut.length - 1]);
  hossz += tavolsag(utolso, elso);
  return hossz;
}
```

```
// Populáció létrehozása (véletlen permutációk)
function létrehozPopulacio(raktarak, meret) {
  const populacio = [];
  const nevek = raktarak.map(r => r.nev);

  for (let i = 0; i < meret; i++) {
    const egyed = [...nevek];

    // Fisher-Yates keverés
    for (let j = egyed.length - 1; j > 0; j--) {
      const rand = Math.floor(Math.random() * (j + 1));
      [egyed[j], egyed[rand]] = [egyed[rand], egyed[j]];
    }
    populacio.push(egyed);
  }
  return populacio;
}

// Populáció rendezése útvonalhossz alapján
function rendez(pop, raktarak) {
  return pop.sort((a, b) => utHossz(a, raktarak) - utHossz(b, raktarak));
}

// Keresztezés
function keresztez(sz1, sz2) {
  const start = Math.floor(Math.random() * sz1.length);
  const end = Math.floor(Math.random() * sz1.length);

  const [min, max] = [Math.min(start, end), Math.max(start, end)];

  const resz = sz1.slice(min, max);
  const gyerek = sz2.filter(g => !resz.includes(g));
  return [...resz, ...gyerek];
}

// Mutáció (két elem felcserélése)
function mutacio(egyed, p) {
  const uj = [...egyed];
  if (Math.random() < p) {
    const i = Math.floor(Math.random() * uj.length);
    const j = Math.floor(Math.random() * uj.length);
    [uj[i], uj[j]] = [uj[j], uj[i]];
  }
  return uj;
}

// Genetikus algoritmus
function genetikusAlgoritmus(raktarak, generaciok = 200, popMeret = 60) {
```

```
let populacio = létrehozPopulacio(raktarak, popMeret);

for (let gen = 0; gen < generaciok; gen++) {

  // Szelekció: a populáció rendezése
  populacio = rendez(populacio, raktarak);

  const uj = [];

  // Populáció feltöltése keresztezéssel és mutációval
  while (uj.length < popMeret) {
    const sz1 = populacio[Math.floor(Math.random() * (popMeret / 2))];
    const sz2 = populacio[Math.floor(Math.random() * (popMeret / 2))];

    let gy = keresztez(sz1, sz2);
    gy = mutacio(gy, 0.2);

    uj.push(gy);
  }

  populacio = uj;
}

// Legjobb megoldás kiválasztása
const legjobb = rendez(populacio, raktarak)[0];
const tav = utHossz(legjobb, raktarak).toFixed(2);

return { legjobb, tav };
}
```